Session IV

System Software Technology

Thomas Gross
Carnegie Mellon University

David Riss
Intel Corporation

Session IV

System Software Technology

Research Concepts

Thomas Gross
Carnegie Mellon University

Scope

iWarp can be used in three different scenarios:

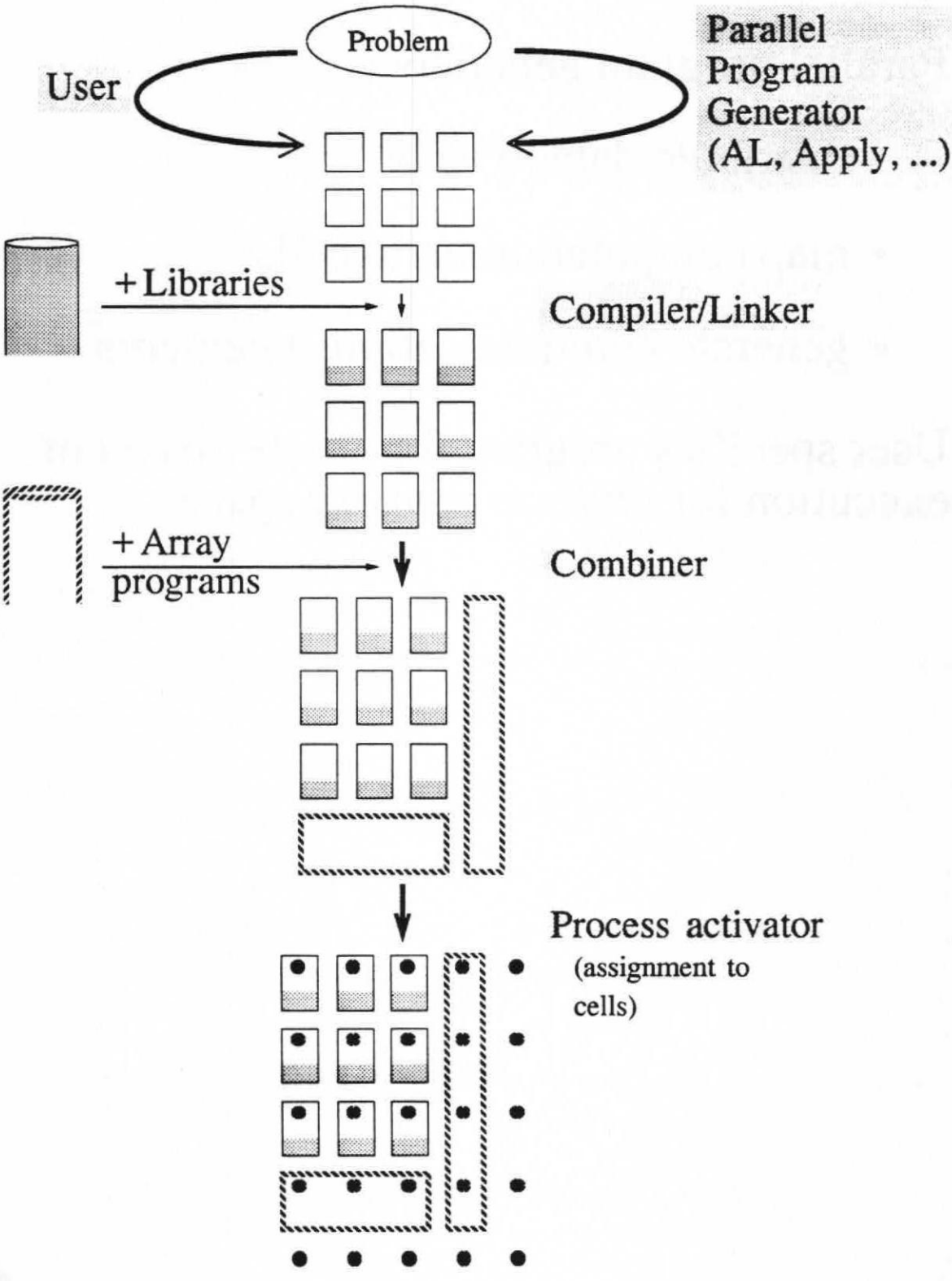
- Large Arrays
  (32+ cells)
- Single-board arrays
  (SBAs) (1-32 cells)
- Application-specific arrays
  (No system software support)

How is iWarp programmed?

Two approaches:

- Parallel program generators
  - Apply
  - AL
- Conventional methods
  - Familiar languages
  - Cell to cell communication operations

How is iWarp programmed?



Parallel program generators

Parallel program generators

- distribute data to cells
- map computation on to cells
- generate communication statements

User specifies program for *single thread* of execution for *uniform memory* space

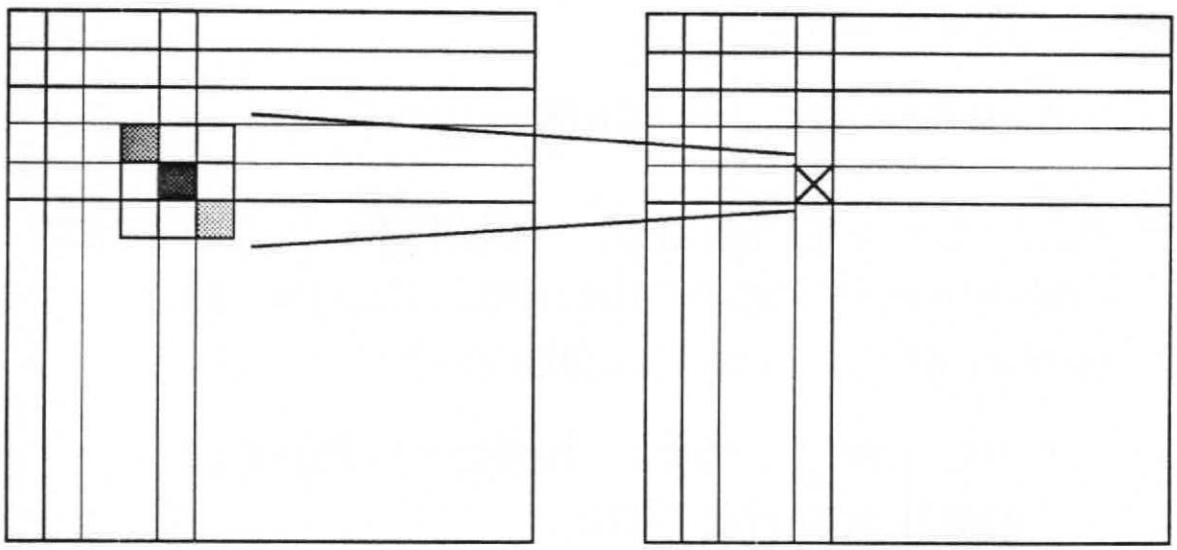
Parallel program generators

Parallel Program generators developed at Carnegie Mellon

- Apply: special-purpose language for image processing
  - deals with borders, image expansion, reduction
  - in use for 3+ years
- AL: array language prototype for domain decomposition problems, (matrix and linear algebra applications)
  - use programmer hints on how to partition the data
  - Automatic mapping of computation
  - in use for 1+ years

Apply

outI := inI(-1,1)+inI(0,0)+inI(1,-1)/3



Input (inI)

Output (outI)

Apply

Apply has been used for low-level vision algorithms such as:

- Color conversion
- Contrast Enhancement
- Edge detection
- Image addition, subtraction, multiplication
- Point operators
- Thresholding

AL

AL compiler maps computation and data onto multiple cells

Key concepts:

- Distributed arrays (**`DARRAY`**)
  - User directive that array *can* be distributed
  - User controls *how* data is distributed
- Normal variables
  - Cannot be distributed
  - Duplicated on all cells
- Partition **`DO`** loops

AL

- Data relations
  - Window relation: array elements must be grouped together
  - Cross relation: different arrays must be grouped together

AL - Example

```
Matrix_Mult(A, B, C, N)
DARRAY float A[][], B[][],
                C[][];
int i, j, k, N;
float Brow[];
XREL ((A,1) (C,1)

do (k=0, n)
ALIGN* (j=0, n)
    Brow[j] = b[k][j];
DO* (i=0, n)
 DO (j=0, n)
 C[i][j] = C[i][j] +
        A[i][j]*Brow[j];
```

Compiler technology

- Developed optimizing compiler for Warp
  - Software pipelining: effective code scheduling technique
- Used compiler in micro-architecture tradeoff studies

Software pipelining

```
FOR (I=0; I<N, I++)
  A[I]=B[I]*C;
```

```
      load C -> R0
      load #0 -> R1
      branch L1
L0:   load B[R1] -> R2
      mul R2, R0 -> R3
      store R3 -> A[R1]
      add #1, R1 -> R1
L1:   bra_cond (R1<N) L0
```

```
load C -> R0
load #0 -> R11
load #0 -> R1
load B[R1++] -> R2
mul R2, R0 -> R3        load B[R1++] -> R2
loop (N-2)
store R3 -> A[R11++]  mul R2, R0 -> R3      load B[R1++] -> R2
                      store R3 -> A[R11++]  mul R2, R0 -> R3
                                            store R3 -> A[R11++]
```

Combiner

Allows the programmer to build an array application

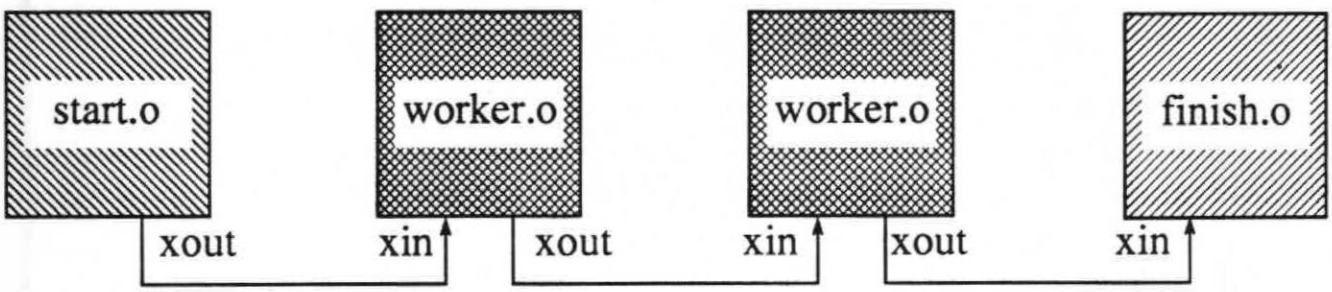
- Creates "array module"
- Homogeneous array programs
- Heterogeneous array programs

Invoked automatically by parallel program generators

Combiner Example

```
for (i=0; i<NUM, i++)
 connect_port(
    Cell[0,i].xout,
    Cell[0,i+1].xin)

Activate(
     Cell[0,0], start.o)
for (i=1, i<NUM, i++)
 Activate(
     Cell[0,i], worker.o)
Activate(Cell[0,NUM],
          finish.o)
```



Execution environment

Host:

Provides external access to the iWarp array

- Workstation
- File server
- Special I/O systems (bulk memory)
- Sensors

Host resident:

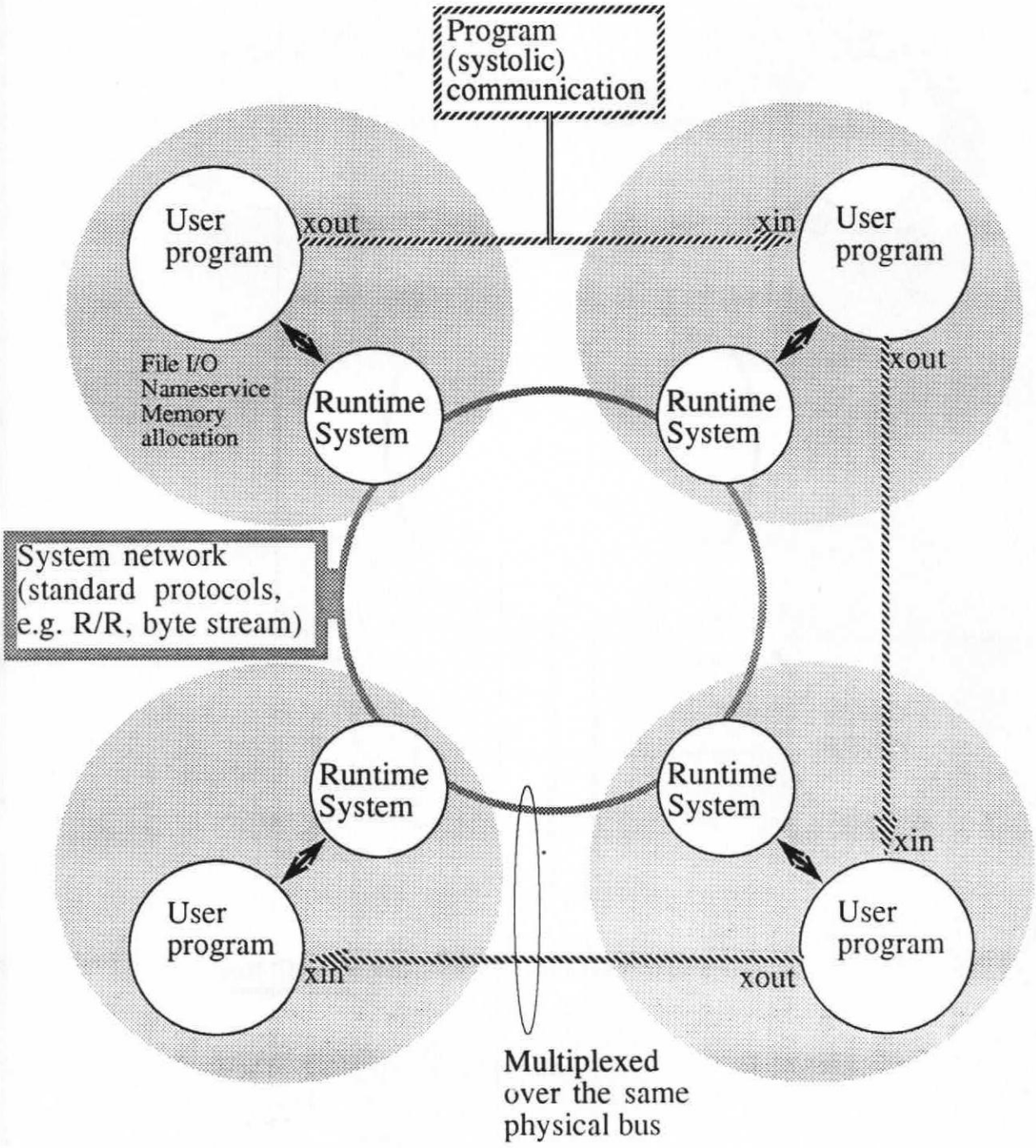
- Array monitor: grants access, supplies data and code, retrieves data, and terminates execution

Execution environment (continued)

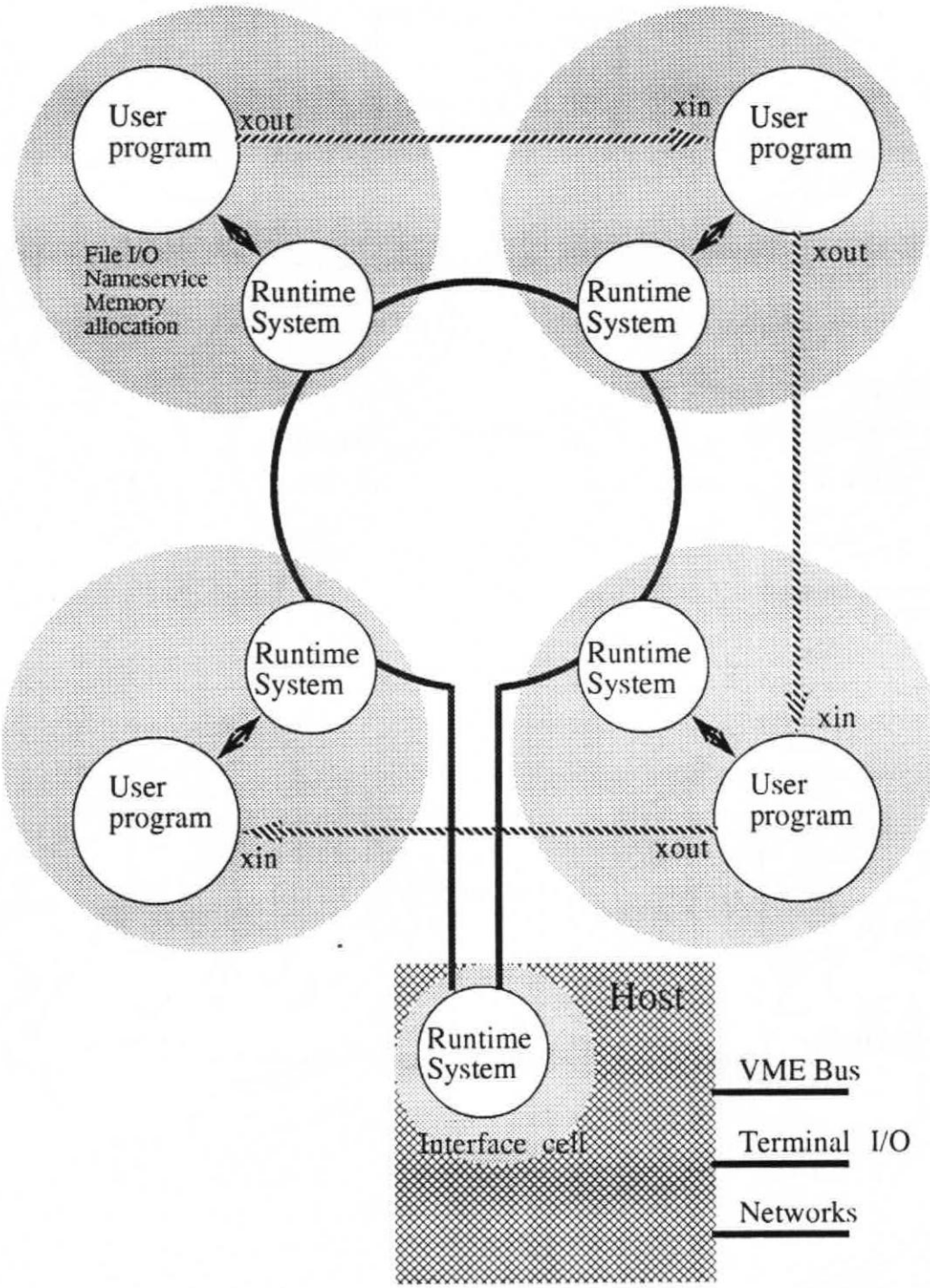
Cell resident:

- Runtime system:
  - down/up loads code and data
  - starts/stops execution
  - interfaces to debugger
  - resource management
  - system network communication

Cell runtime system



Host interaction



Session IV

System Software Technology

Delivered Capability

David Riss
Intel Corporation

Presentation Overview

- Basic software: outline
  - Single cell issues
  - Array issues
- Program execution: outline
- Basic software: details
- Program runtime system: details
- Summary
- Questions & answers

Program Development Environment

Contains the software building blocks to develop application programs for iWarp systems

Uses operating system of development platform to supply:

- editors
- version control
- window management

Identical software environment for single board array or processor array

Program development software executable on any node in the LAN

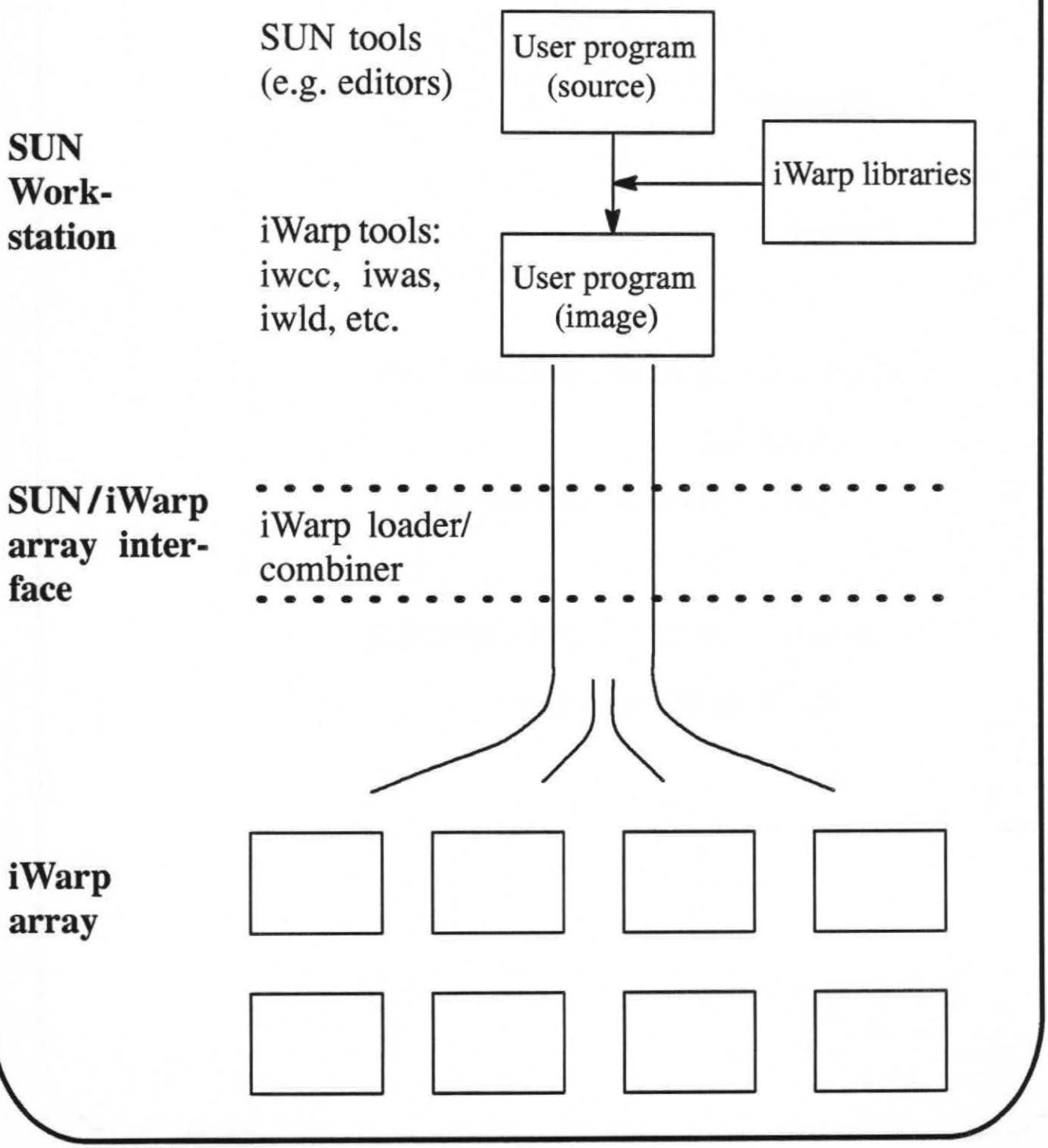
iWarp Single–Cell Program Development Tools

- **C Optimizing Compiler**
  - standard Kernighan & Ritchie C
  - software pipelining (long instruction word optimizations)
  - direct access to system level communication primitives (send & receive); h/w conditions
  - assembly code inlining facility
- **FORTRAN Optimizing Compiler**
  - standard ANSI 77 FORTRAN
  - VAX/VMS extensions
  - direct access to system level communication primitives (send & receive); h/w conditions
  - Inlining of compiled C functions in FORTRAN programs
  - Shared back–end with C compiler
- **Common Calling Conventions**

iWarp Single–Cell Program Development Tools

- **Assembler**
- **Linker**
- **Object Code Manipulation Tools**
  - librarian
  - object code cross reference
- **Symbolic Source Level Debugger**
  - built-in disassembler

Program Development Environment



Program Development Environment

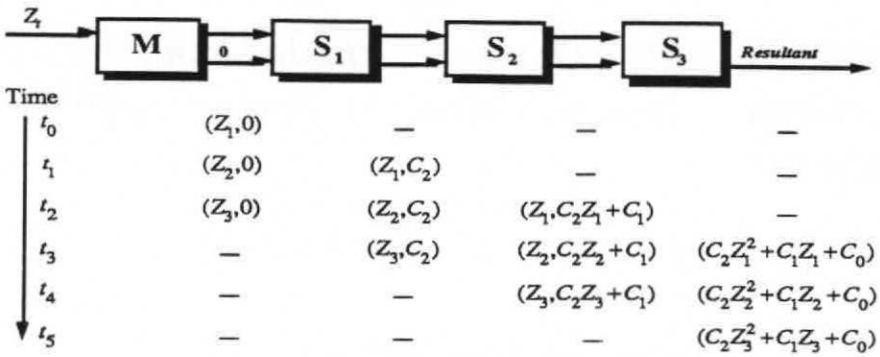
Standard direct file I/O between the host and the iWarp array permit simple and straight forward application development.

Steps in developing an application.

- Write and debug the application on the host (using the host I/O).
- Move the application to a single cell array (using the host I/O).
- Distribute the application across multiple cells to test and debug communication.
- Tailor the I/O to the application, removing or modifying host file I/O as desired.

Program Development Environment

$$p(z) = c_n z^n + \ldots + c_1 z + c_0$$




```
for ( i = 0; i < nz; i++ ) {
  _sendf( GATE0, z[i] );
  _sendf( GATE1, fzero );
}
```

```
for ( i = 0; i < nz; i++ ) {
  xin = _receivef( GATE0 );
  yin = _receivef( GATE1 );
  _sendf( GATE0, xin );
  _sendf( GATE1, coeff + yin * xin );
}
```

```
for ( i = 0; i < nz; i++ ) {
  ftmp = _receivef( GATE0 );
  p[i] = _receivef( GATE1 );
}
```

- Input initial values of coefficients and solution points (z) from host using **scanf**.
- Using systolic communication provided through the **send** and **receive** intrinsic functions, the terms of the polynimial are accumulated.
- Final values of **p** are collected and sent to host using **printf**.

Program Development Environment

```
#include <stdio.h>

asm float fast_dp(addrp,addrq,cnt)
{
% register addrp,addrq; tmpreg cnt,t1,t2,t3,t4; tmpreg return;
   movereg.w addrp,t1
   movereg.w addrq,t2
   sub 2,cnt
   {                                          ld.w (t1,4),lm2;    ld.w (t2,4)+=,lm0}
   {                                          ld.w (t1,4)+=,lm2; ld.w (t2,4)+=,lm0}
   {                    fmul.f lm0,lm2,t3;    ld.w (t1,4)+=,lm2; ld.w (t2,4)+=,lm0}

loop cnt

el{
    fadd.f t4,t3,t4;    fmul.f lm0,lm2,t3;    ld.w (t1,4)+=,lm2; ld.w (t2,4)+=,lm0
   }
   {fadd.f t4,t3,t4;    fmul.f lm0,lm2,t3;    ld.w (t1,4)+=,lm2}
   {fadd.f t4,t3,t4;    fmul.f lm0,lm2,t3}
   {fadd.f t4,t3,return}
}

main()
{
  register float *p = &a[0];
  register float *q = &b[0];
  register int vec_len;
  register float s = 0.0;
  register int i;

/*  for(i=0;i<vec_len;i++){
 *    s = s + a[i]*b[i];
 *  }
 */

  s = fast_dp(p,q,vec_len);

}
```

The assembly language inlining feature provides a powerful method of including specific assembly language segments into ordinary C code. Parameters to the pseudo–function are associated with registers allocated by the compiler to provide a very efficient and flexible interface to the assembly sequence. Here a single C&A instruction is used to perform a dot–product. The software–pipelining optimization phase of the C compiler will eventually provide similarly efficient code from appropriate C code.

Application Program Development Tools

Apply

- an image processing language
- callable by standard C and FORTRAN
- WEB library with over 100 routines
- generates C source code

Apply is the first step in parallel program generators for iWarp

Program Runtime Environment

iWarp Runtime System (host-resident)

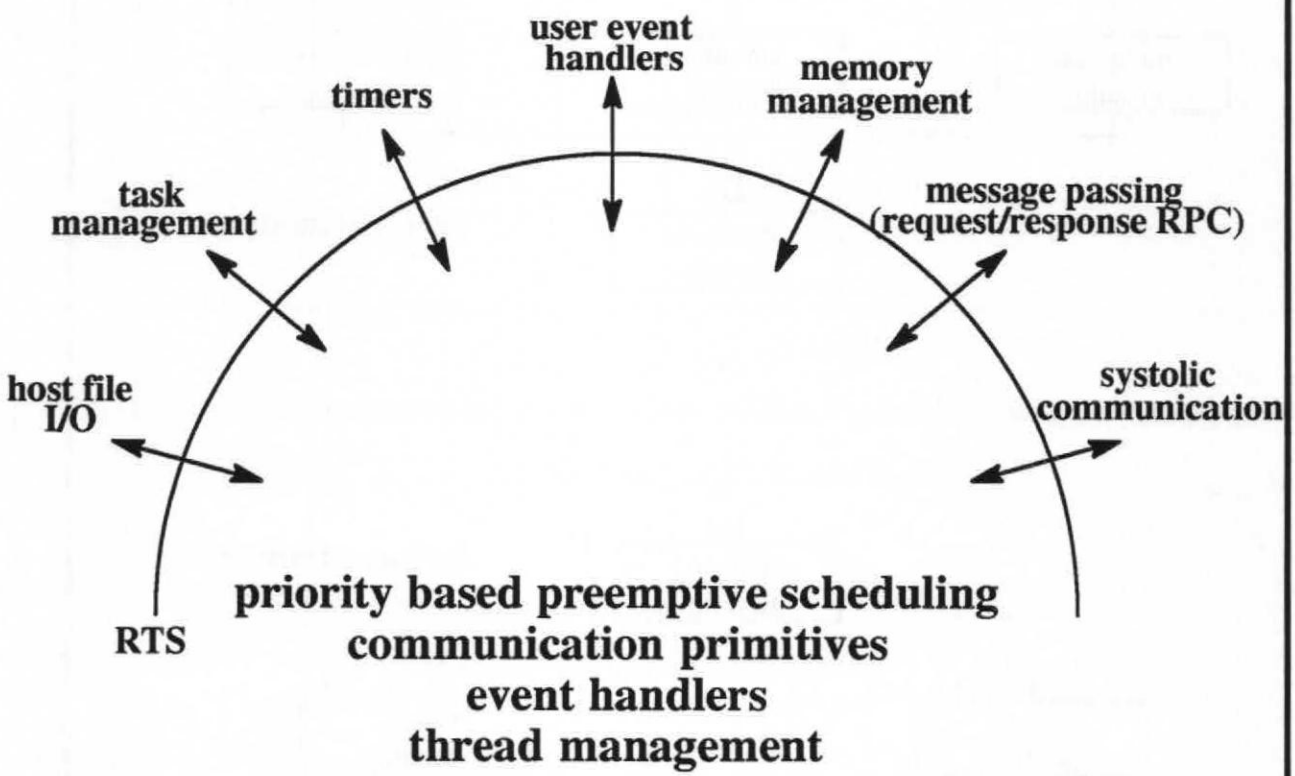
- **Combiner**
  - joins multiple cell programs into a single array load module
- **Array Allocation**
  - grants access to iWarp hardware
  - queues jobs awaiting execution
- **Array Job Management**
  - job activation/kill
  - obtain job state information
  - multi-cell debugger

Program Runtime Environment

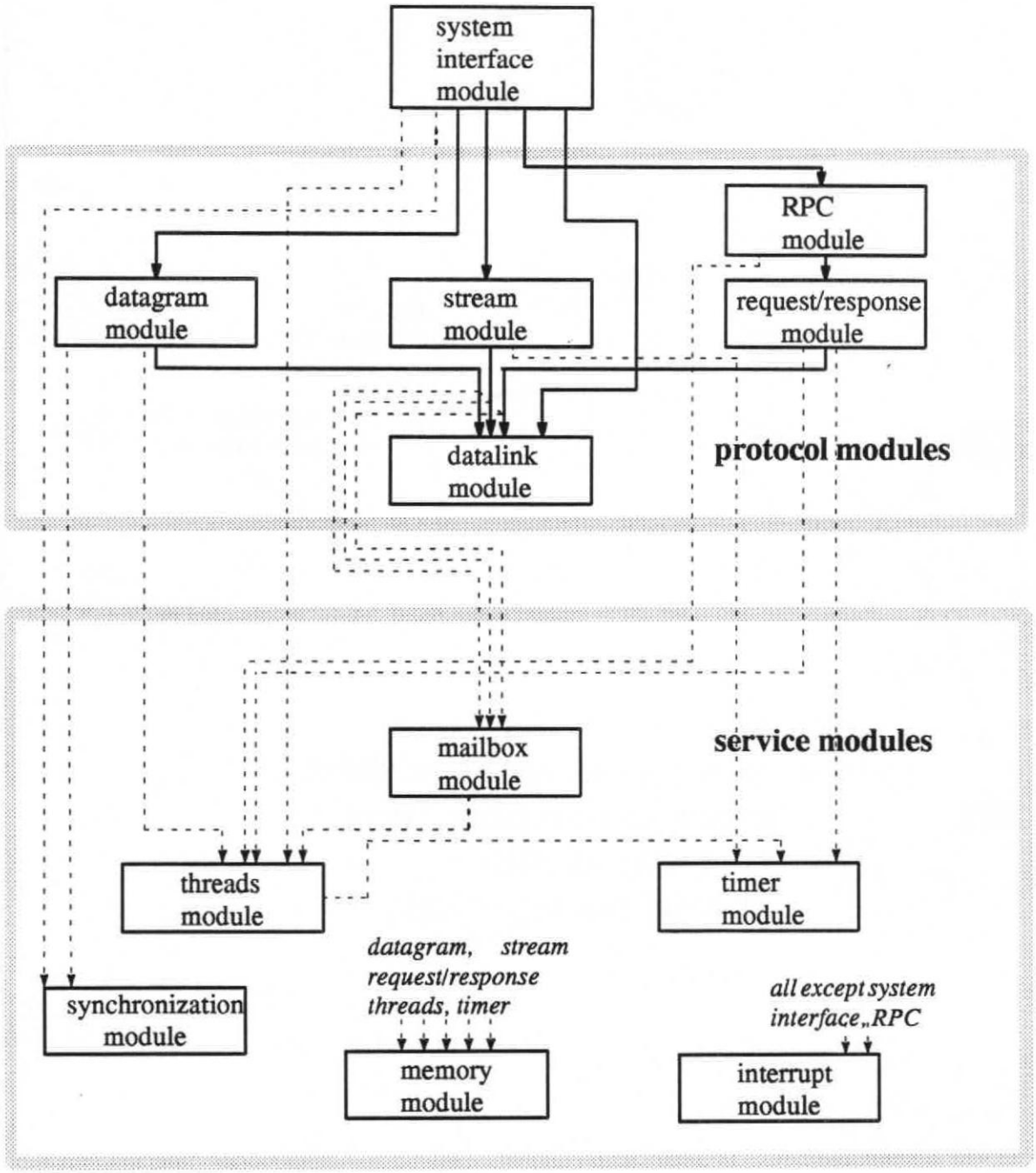
iWarp Runtime System (cell-resident)

- **Basic Runtime Kernel**
  - interrupt handlers
  - thread management
  - resource management
- **Process Activator**
  - loads user data and code
  - starts application program execution on cell
- **User-programmed Communication**
  - memory–to–memory message passing
  - systolic communication
- **Standard Libraries**
  - UNIX System V standard I/O
  - iPSC compatibility library
  - math intrinsic function library

Program Runtime Environment



iWarp Runtime System



Summary

- **Complete set of program development tools**
  - Optimizing compilers
  - Parallel program generators
  - Debugger
- **Runtime environment for efficient execution**
  - Systolic communication
  - Well-known message-based protocol types (request/response, RPC)
- **Base for effective problem partitioning**
  - Large arrays (1+ GFLOPS)
  - Single-board arrays